Programmation et IA

Cours 3

Jean-Jacques Lévy

jean-jacques.levy@inria.fr

http://jeanjacqueslevy.net/prog-ia-25

Plan

- récursivité
- graphique avec la petite tortue
- module des nombres aléatoires
- génération de listes avec compréhension
- révision du tri par sélection

[texte des programmes]

Pour apprendre Python:

w3schools: tutoriel et réference

`https://www.w3schools.com/python/default.asp`

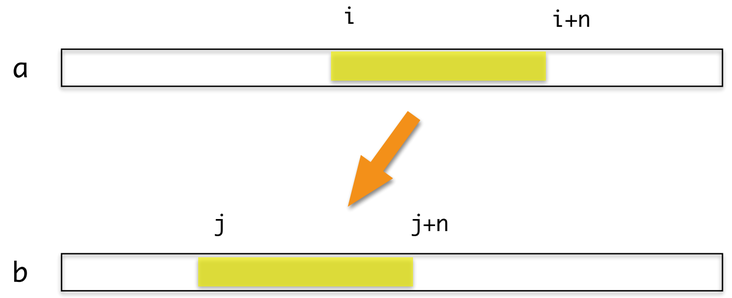
programiz: tutoriel et réference

`https://www.programiz.com/python-programming`

Valeur d'un tableau — Alias

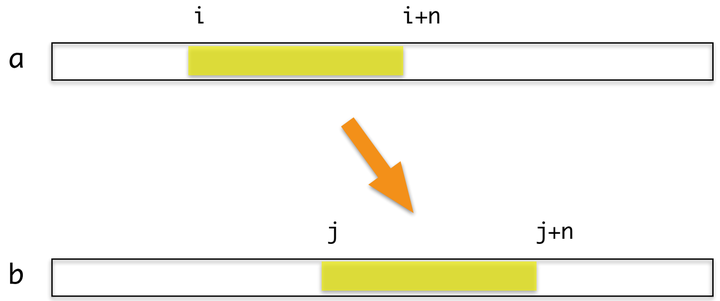
Exercice 1: le programme suivant est-il correct?

```
def copy (a, b, i, j, n) :
    for k in range (n) :
        b [j + k] = a [i + k]
```



Solution: Non! il faut écrire le programme suivant, correct même quand les paramètres *a* et *b* sont des alias.

```
def copy (a, b, i, j, n) :
        if i > j :
            for k in range (n) :
                b [ j + k ] = a [i + k]
        else :
            for k in range (n-1, -1, -1) :
                b [ j + k ] = a [ i + k]
```



[texte des programmes]

récursivité

Fonctions récursives

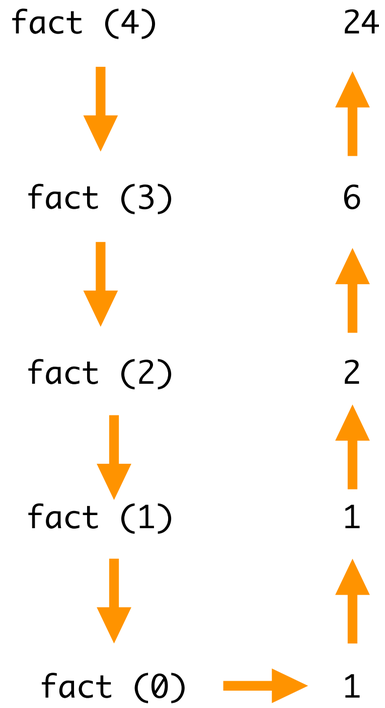
- une fonction qui se rappelle avec un argument plus petit

```
def fact (n) :
    if n == 0 :
        return 1
    else :
        return n * fact (n-1)
```

factorielle

```
>>> fact(3)
6
>>> fact(4)
24
>>> fact(10)
3628800
```

appels récursifs




```
fact (4) == 4 * fact (3) == 4 * 3 * fact (2) == 4 * 3 * 2 * fact (1) == 4 * 3 * 2 * 1 * fact (0) == 4 * 3 * 2 * 1 * 1
```

[texte des programmes]

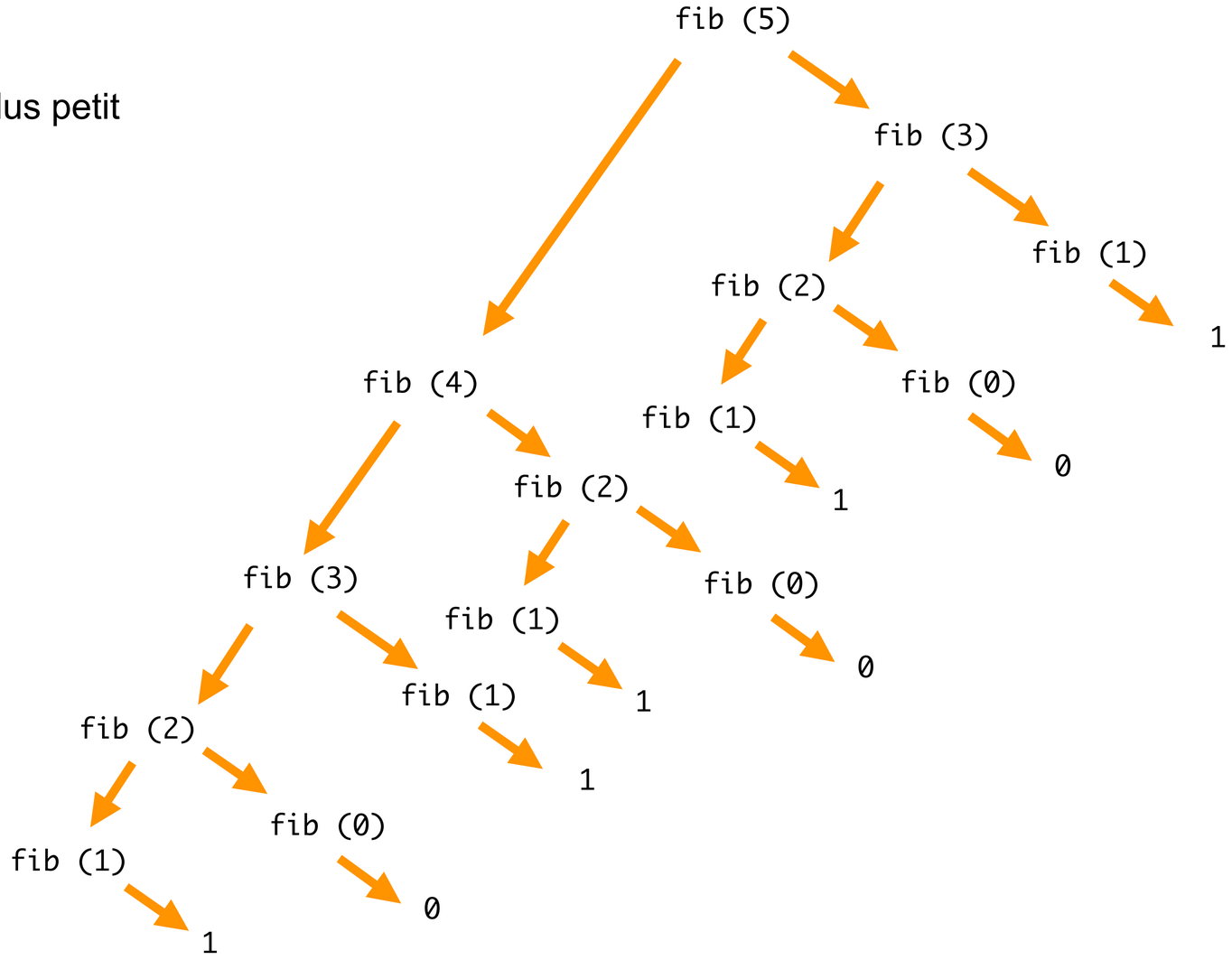
Fonctions récursives

- une fonction qui se rappelle avec un argument plus petit

```
def fib (n) :
    if n <= 1 :
        return n
    else :
        return fib (n-1) + fib (n-2)
```

fibonacci

```
>>> fib (10)
55
>>> fib (20)
6765
>>> fib (35)
9227465
```



Exercice 2: écrire une fonction qui calcule `fibonacci` plus rapidement

[texte des programmes]

Fonctions récursives

- une fonction qui se rappelle avec un argument plus petit ?

```
def f (n) :
    if n > 100 :
        return n - 10
    else:
        return f(f(n+11))
```

Exercice 3: quelle est la valeur de cette fonction ?

```
f (103) == 93
f (102) == 92
f (101) == 91
f (100) == f (f(111)) == f (101) == 91
f (99) == f (f(110)) == f (100) == .. 91
f (98) == f (f(109)) == f (99) == .. 91
f (97) == f (f(108)) == f (98) == .. 91
f (96) == f (f(107)) == f (97) == .. 91
...
f (91) == f (f(102)) == f(92) == .. 91
f (90) == f (f(101) == f (91) == .. 91
f (89) == f (f(100)) == .. f (91) == .. 91
f (88) == f (f(99)) == .. f (91) == .. 91
...
```

[texte des programmes]

Fonctions récursives

- la fonction d'Ackermann croit très vite !

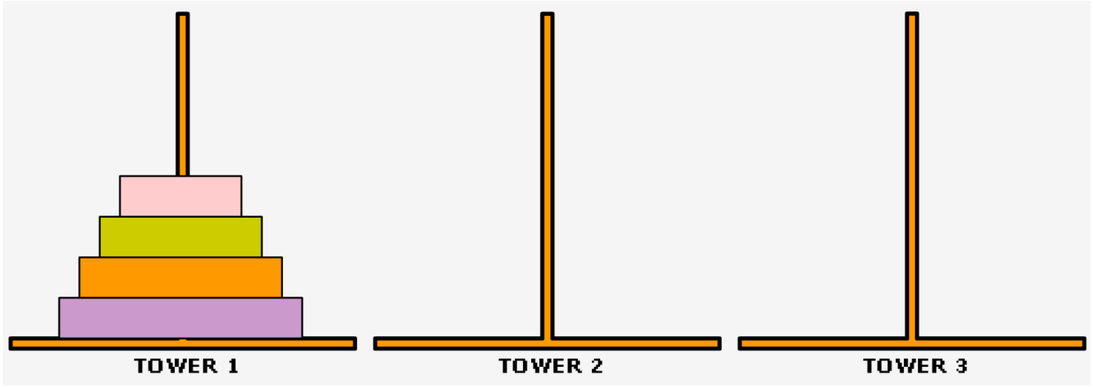
```
def A (m, n) :
    if m == 0 :
        return n + 1
    elif n == 0:
        return A (m-1, 1)
    else :
        return A (m-1, A (m, n-1))
```

Valeurs de $A(m, n)$

$m \backslash n$	0	1	2	3	4	n
0	1	2	3	4	5	$n + 1$
1	2	3	4	5	6	$n + 2$
2	3	5	7	9	11	$2n + 3$
3	5	13	29	61	125	$2^{n+3} - 3$
4	13	65533	$2^{65536} - 3$	$A(3, 2^{65536} - 3)$	$A(3, A(4, 3))$	$2^{2^{\cdot^{\cdot^{\cdot^{2}}}}} - 3$ ($n + 3$ termes)
5	65533	$A(4, 65533)$	$A(4, A(5, 1))$	$A(4, A(5, 2))$	$A(4, A(5, 3))$	
6	$A(5, 1)$	$A(5, A(5, 1))$	$A(5, A(6, 1))$	$A(5, A(6, 2))$	$A(5, A(6, 3))$	

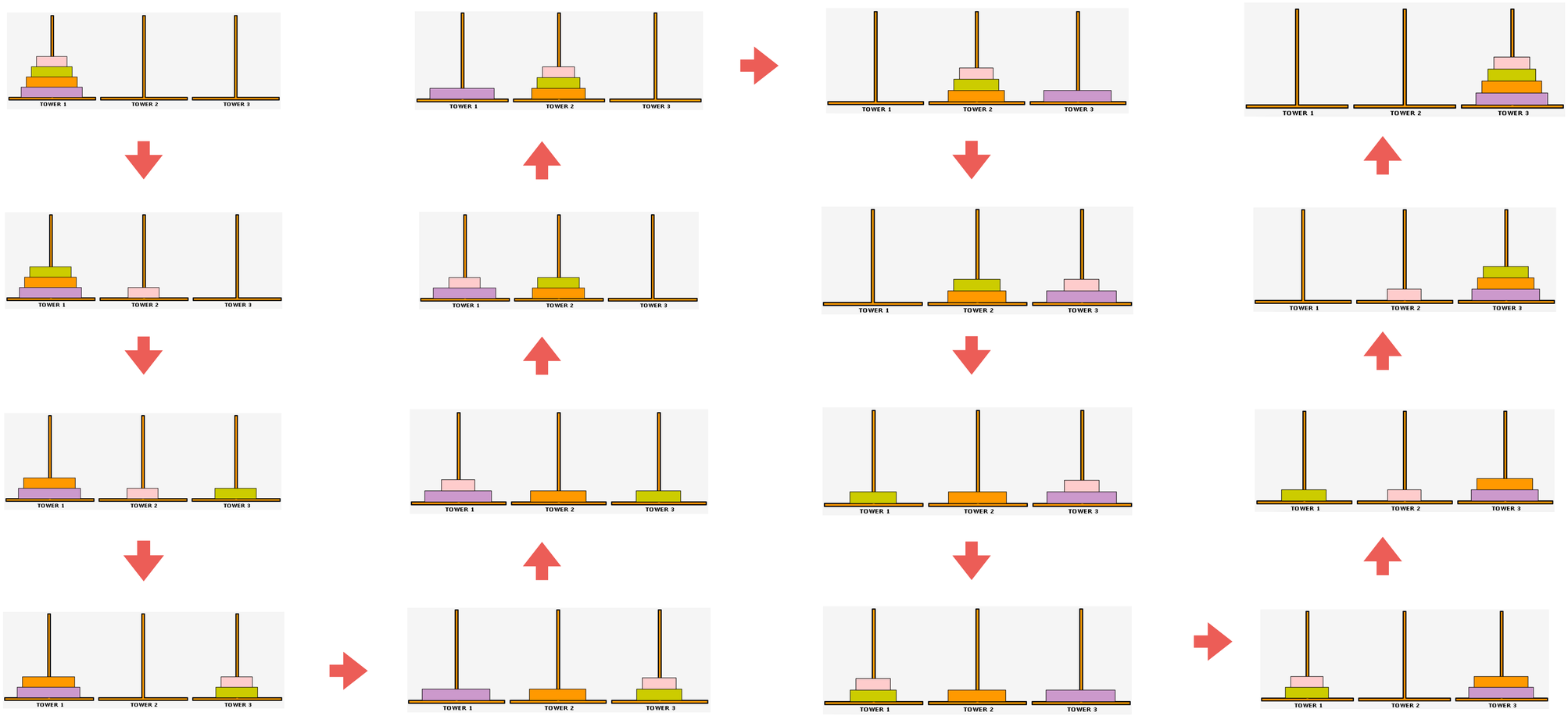
[texte des programmes]

Les tours de Hanoi



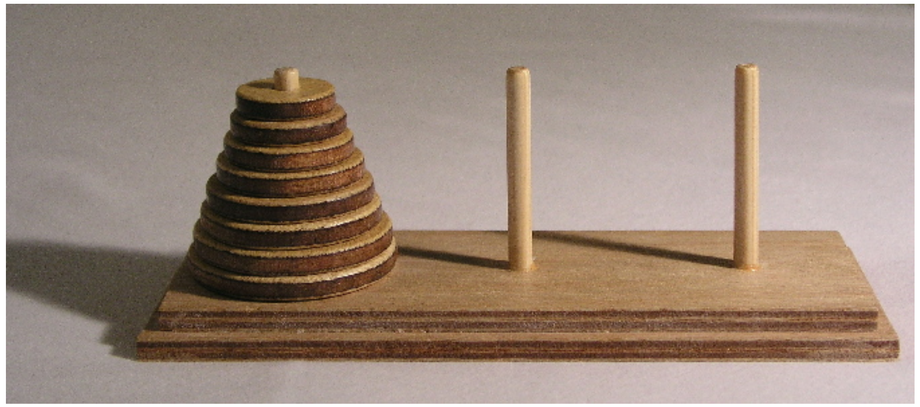
jamais petit
sous un grand

Les tours de Hanoi



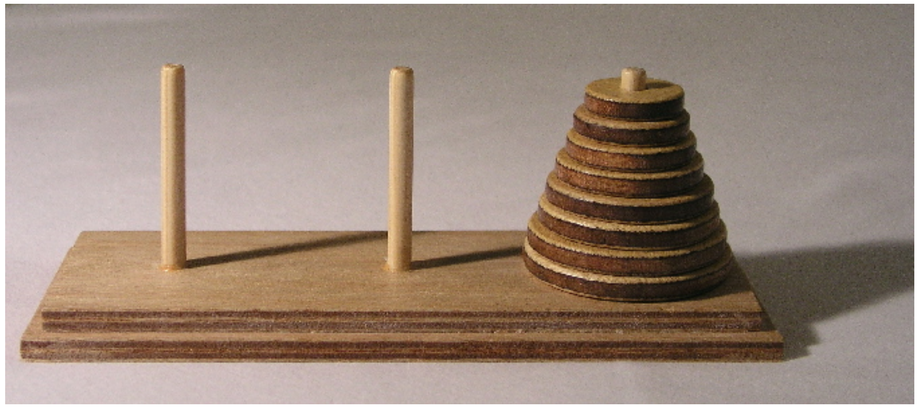
Les tours de Hanoi

- on a 3 piles et n rondelles sur la pile 1
- jamais une rondelle grosse au-dessus d'une rondelle petite



pile 1 pile 2 pile 3

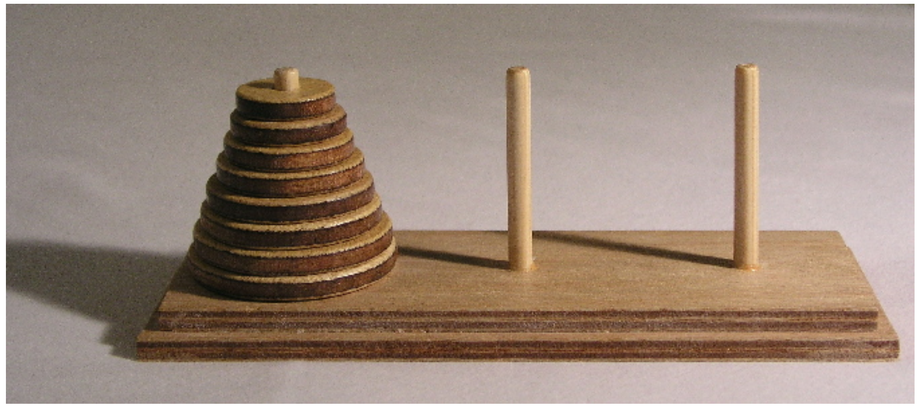
- il faut amener les n rondelles sur la pile 3
- on ne déplace qu'une seulle rondelle à la fois
- et on ne met jamais une rondelle au-dessus d'une plus petite



pile 1 pile 2 pile 3

Les tours de Hanoi

- on a 3 piles et n rondelles sur la pile 1
- jamais une rondelle grosse au-dessus d'une rondelle petite



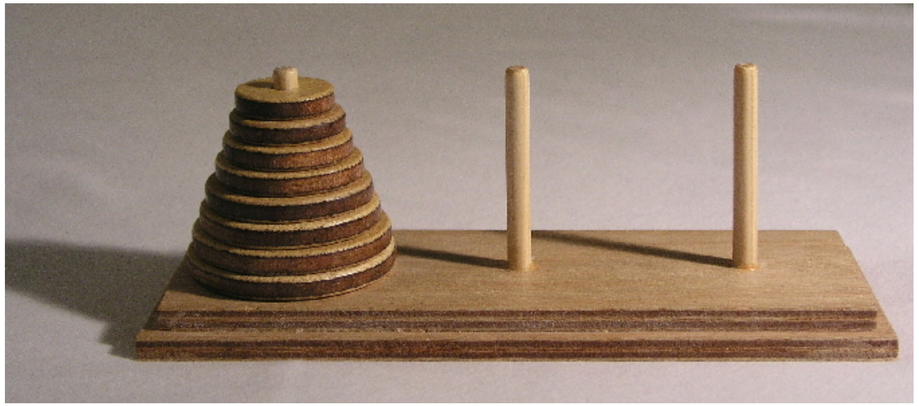
pile 1 **pile 2** **pile 3**

- il faut amener les n rondelles sur la pile 3
- on ne déplace qu'une seule rondelle à la fois
- et on ne met jamais une rondelle au-dessus d'une plus petite

Les tours de Hanoi

- on généralise le problème pour aller de la pile i à la pile j
  où $1 \leq i \leq 3$ et $1 \leq j \leq 3$

  la troisième pile est alors $6 - i - j$



pile i pile $6 - i - j$ pile j

- supposons le problème résolu pour n-1 rondelles entre pile i et pile j

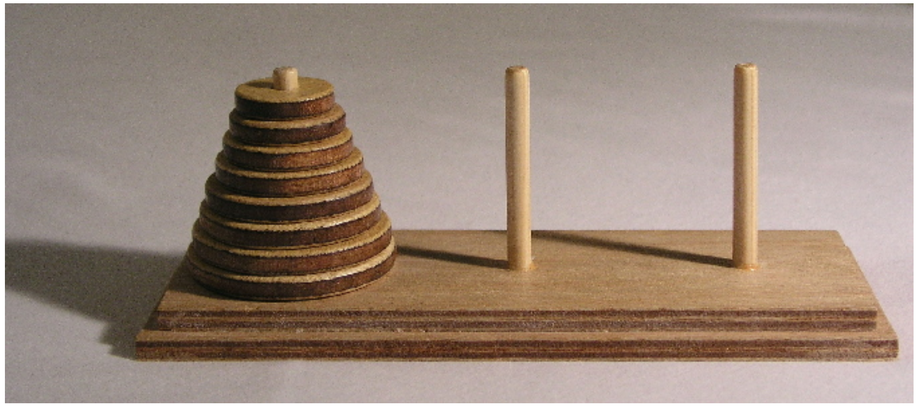
- j'amène les n-1 rondelles du dessus de la pile i sur la troisième pile
- j'amène la grosse rondelle de la pile i vers la pile j
- j'amène les n-1 rondelles de la troisième pile vers la pile j

```
def hanoi (n, i, j) :
    if n > 0 :
       hanoi (n-1, i, 6 - i -j)
       print (f"{i} --> {j}")
       hanoi (n-1, 6 - i - j, j)
```

Les tours de Hanoi

```
>>> hanoi (2, 1, 3)
1 --> 2
1 --> 3
2 --> 3
>>> hanoi (3, 1, 3)
1 --> 3
1 --> 2
3 --> 2
1 --> 3
2 --> 1
2 --> 3
1 --> 3
>>> hanoi (4, 1, 3)
1 --> 2
1 --> 3
2 --> 3
1 --> 2
3 --> 1
3 --> 2
1 --> 2
1 --> 3
2 --> 3
2 --> 1
3 --> 1
2 --> 3
1 --> 2
1 --> 3
2 --> 3
```

```
>>> hanoi (5, 1, 3)
1 --> 3
1 --> 2
3 --> 2
1 --> 3
2 --> 1
2 --> 3
1 --> 3
1 --> 2
3 --> 2
3 --> 1
2 --> 1
3 --> 2
1 --> 3
1 --> 2
3 --> 2
1 --> 3
2 --> 1
2 --> 3
1 --> 3
2 --> 1
3 --> 2
3 --> 1
2 --> 1
2 --> 3
1 --> 3
1 --> 2
3 --> 2
1 --> 3
2 --> 1
2 --> 3
1 --> 3
```



pile i pile $6 - i - j$ pile j

- les tours de Hanoi sont un exemple du raisonnement inductif

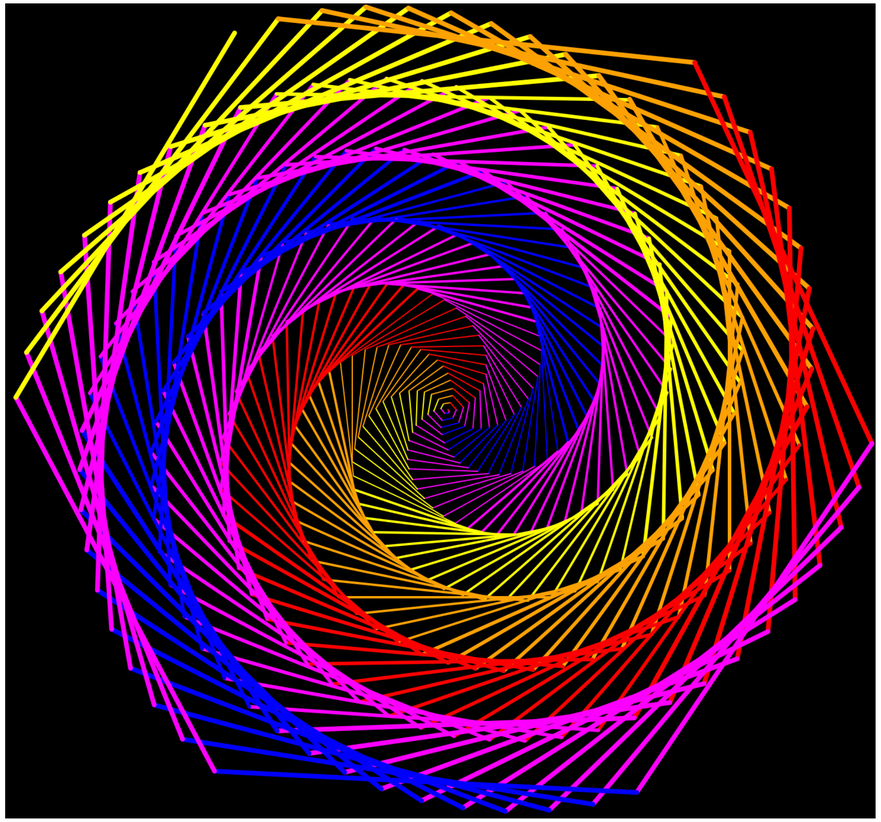
  (aussi appelé raisonnement par récurrence)

récursivité ⟷ raisonnement inductif

Graphique avec la tortue

- un paquetage `turtle.py` simple pour apprendre l'informatique dans les écoles
  (cf. http://fr.wikipedia.org/wiki/Seymour_Papert)

```
import turtle

def spiralArt():
    colors = ['orange', 'red', 'magenta' , 'blue', 'magenta', \
              'yellow', 'green', 'cyan', 'purple']
    turtle.bgcolor('black')
    turtle.speed (0)
    for x in range (360):
        turtle.pencolor(colors [x % 6])
        turtle.pensize (x//100 + 1)
        turtle.forward (x)
        turtle.right (59)
    turtle.hideturtle()
    turtle.done()
```

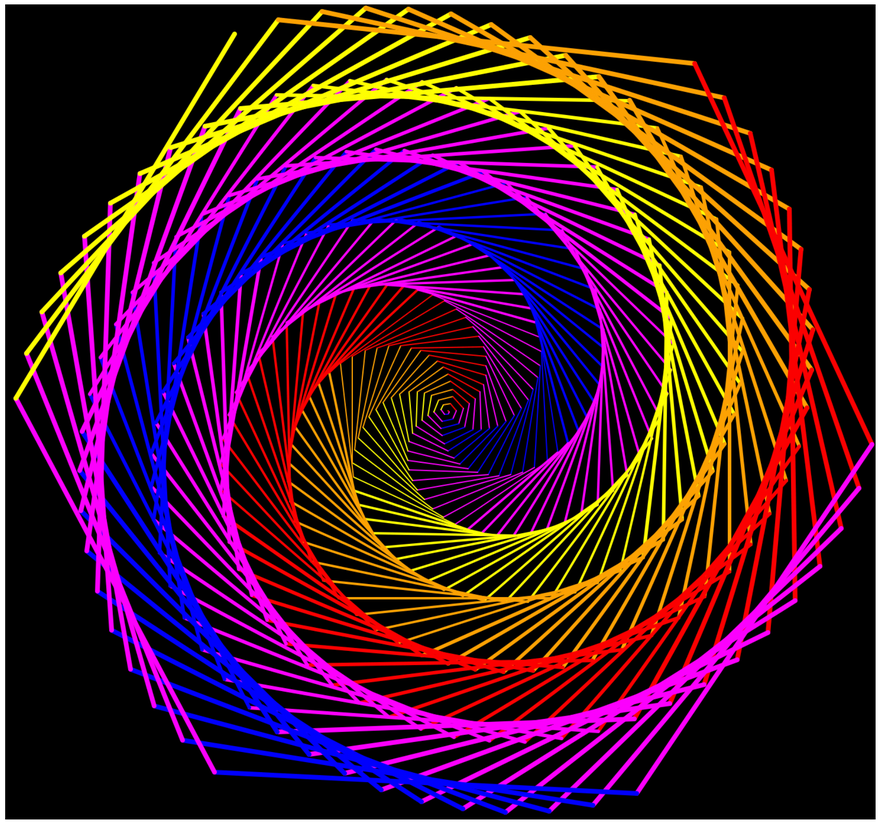


[texte des programmes]

Graphique avec la tortue

- un paquetage `turtle.py` simple pour apprendre l'informatique dans les écoles
  (cf. http://fr.wikipedia.org/wiki/Seymour_Papert)

```
import turtle as t

def spiralArt():
    colors = ['orange', 'red', 'magenta' , 'blue', 'magenta', \
              'yellow', 'green', 'cyan', 'purple']
    t.bgcolor('black')
    t.speed (0)
    for x in range (360):
        t.pencolor(colors [x % 6])
        t.pensize (x//100 + 1)
        t.forward (x)
        t.right (59)
    t.hideturtle()
    t.done()
```



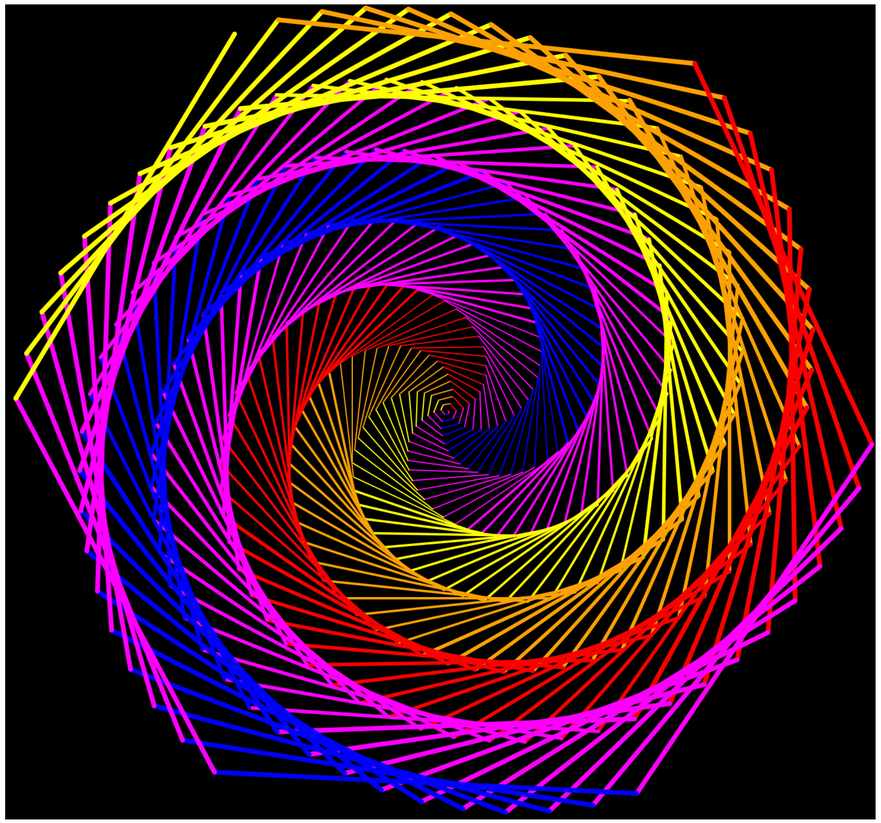
- écriture plus légère

[texte des programmes]

Graphique avec la tortue

- un paquetage `turtle.py` simple pour apprendre l'informatique dans les écoles (cf. http://fr.wikipedia.org/wiki/Seymour_Papert)

```
from turtle import *

def spiralArt():
    colors = ['orange', 'red', 'magenta' , 'blue', 'magenta', \
              'yellow', 'green', 'cyan', 'purple']
    bgcolor('black')
    speed (0)
    for x in range (360):
        pencolor(colors [x % 6])
        pensize (x//100 + 1)
        forward (x)
        right (59)
    hideturtle()
    done()
```



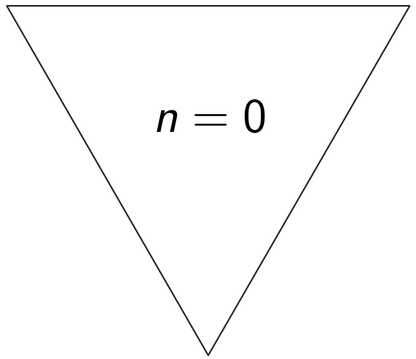
- écriture encore plus légère (mais dangereuse car possibilité de confusion avec identificateurs existants)

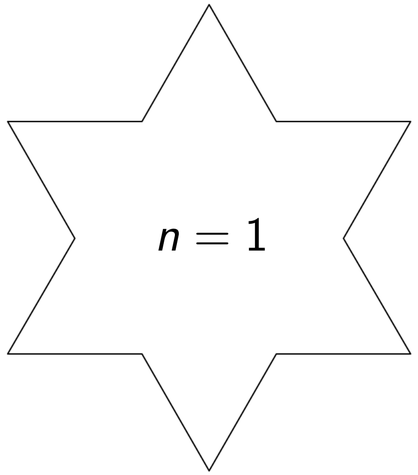
Graphique avec la tortue

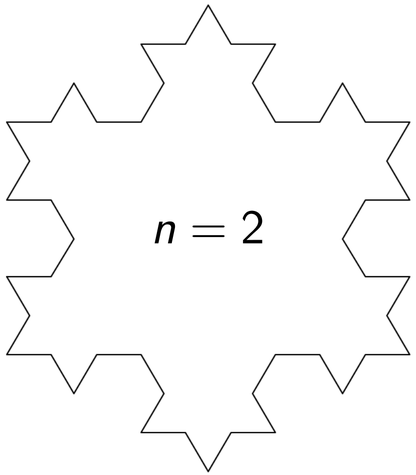
- le flocon de von Koch

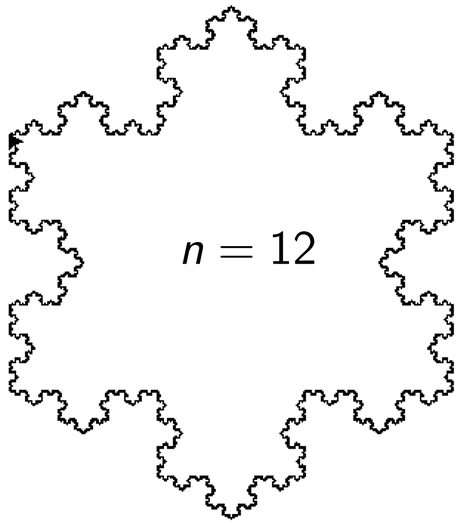
```
def koch (l, n) :
    if n <= 0 :
        t.forward (l)
    else:
        koch (l/3, n-1); t.left(60)
        koch (l/3, n-1); t.right (120)
        koch (l/3, n-1); t.left (60)
        koch (l/3, n-1)

def flocon (l, n) :
    koch (l, n); t.right (120)
    koch (l, n); t.right (120)
    koch (l, n)
```

```
def drawFlocon (length, order) :
    t.speed (100)
    flocon (length, order)
    t.hideturtle()
    t.done ()

drawFlocon (300, 4)
```









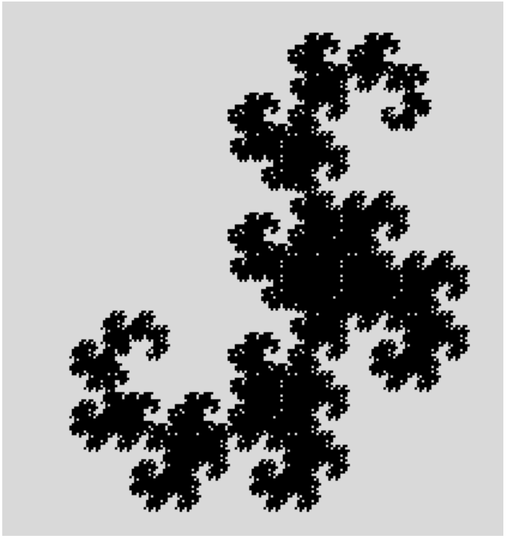
[texte des programmes]

Graphique avec la tortue

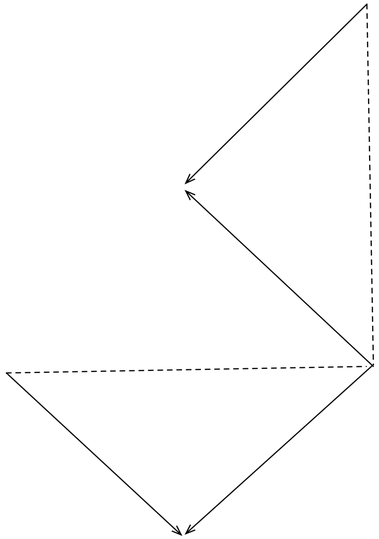
- la courbe du dragon

```
def dragon (n, l, s, color) :
    if n <= 1 :
       t.pencolor (color)
       t.forward (l)
    else :
       t.right (45*s)
       dragon (n-1, l/1.4142, 1, color)
       t.left (90*s)
       dragon (n-1, l/1.4142, -1, color)
       t.right (45*s)
```

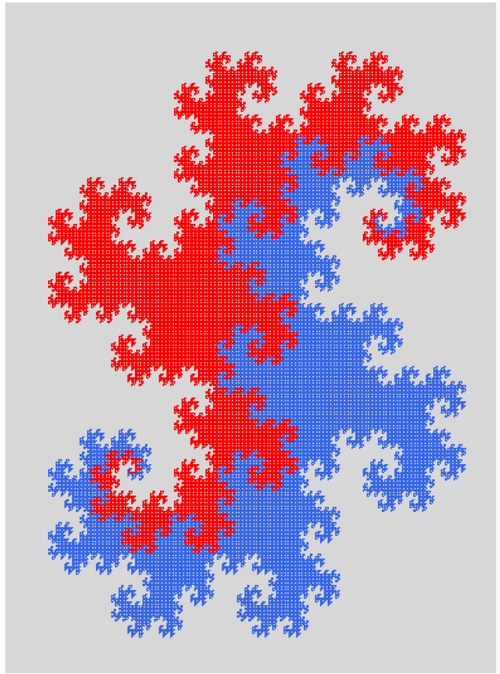
```
def drawDragon (order, length) :
    t.speed (100)
    dragon (order, length, 1, 'black')
    t.hideturtle()
    t.done ()

drawDragon (12, 200)
```



- on plie une feuille de papier n fois



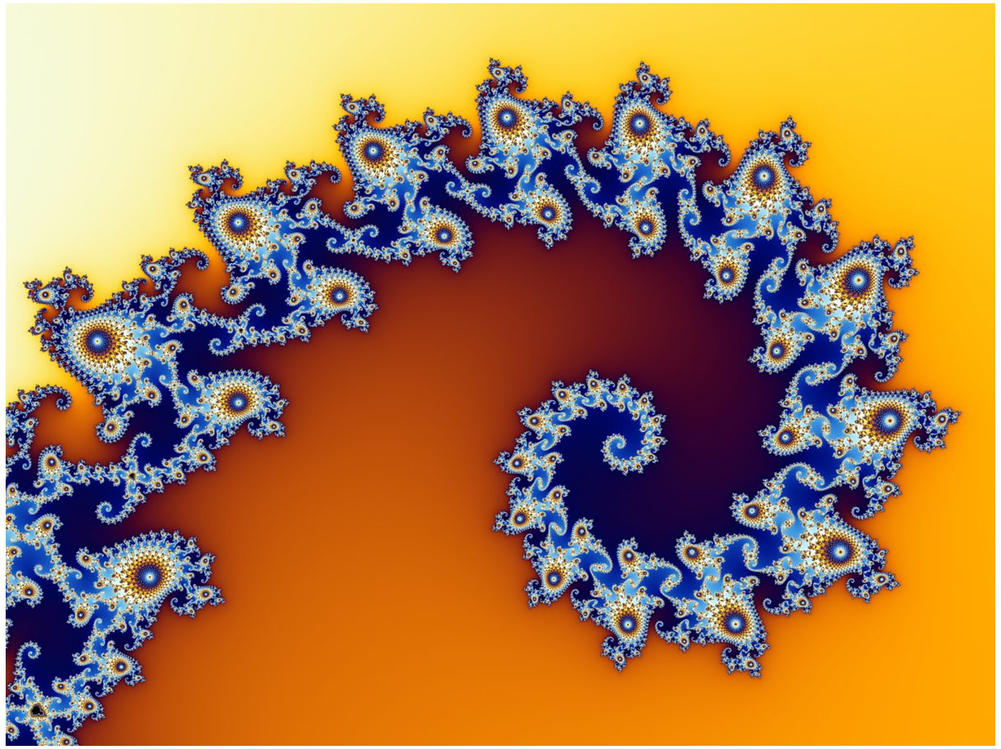
```
def drawTwinDragons (order, length) :
    t.speed (100)
    dragon (order, length, 1, 'blue')
    t.penup(); t.home (); t.pendown()
    dragon (order, length, -1, 'red')
    t.hideturtle()
    t.done ()

drawTwinDragons (13, 200)
```

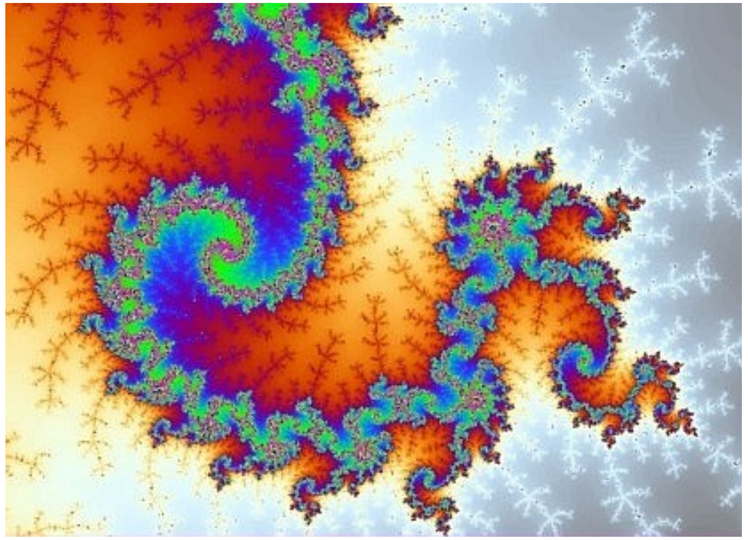


[texte des programmes]

Fractales

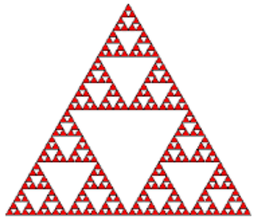
- les fractales de Benoît Mandelbrot

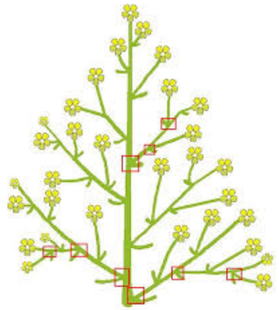


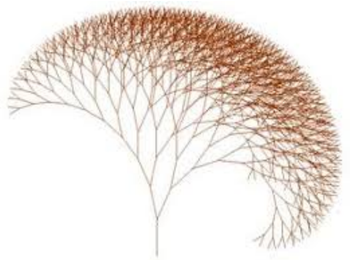


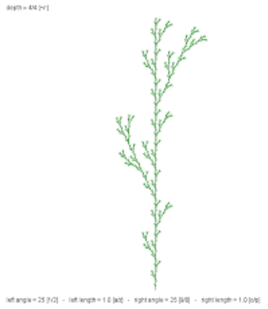
Fractales

- les fractales de Benoît Mandelbrot



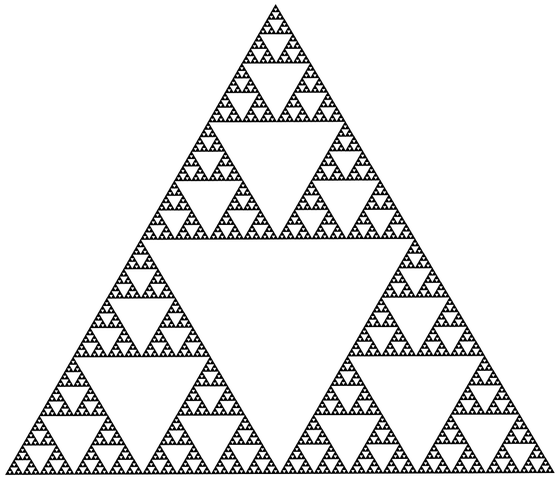




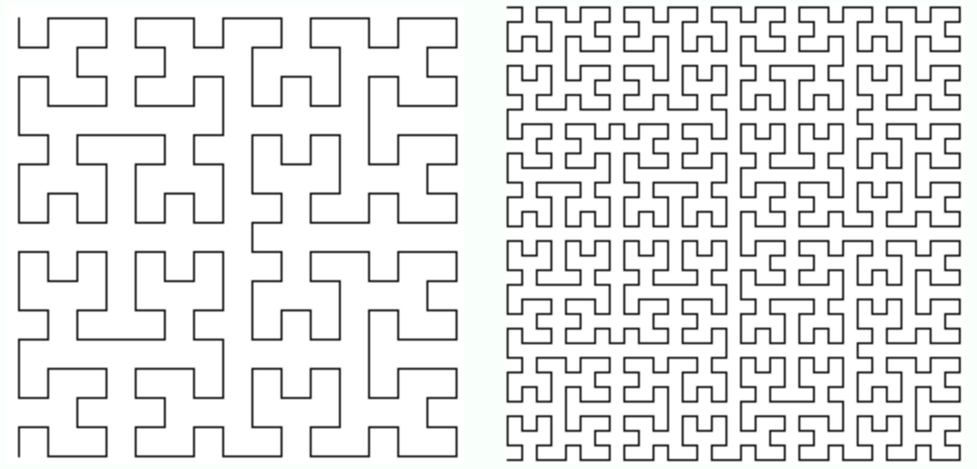


Fractales

Exercice 4: fonctions écrire les programmes pour dessiner les 2 fractales suivantes



sierpinsky



hilbert

compréhension de listes

Modules

- les fonctions ou données (de librairie..) sont regroupées en modules
- par exemple, le module `random` des nombres aléatoires

```
import random as rd
```

- quelques fonctions de ce module:

```
rd.randrange (m, n)
rd.randint (m, n)
rd.choice (s)

rd.random ()

rd.seed (n)
```

nombre aléatoire dans `range(m, n)` et `range(m, n+1)`

nombre aléatoire choisi dans la liste `s`

nombre réel aléatoire choisi dans l'intervalle *[0, 1[*

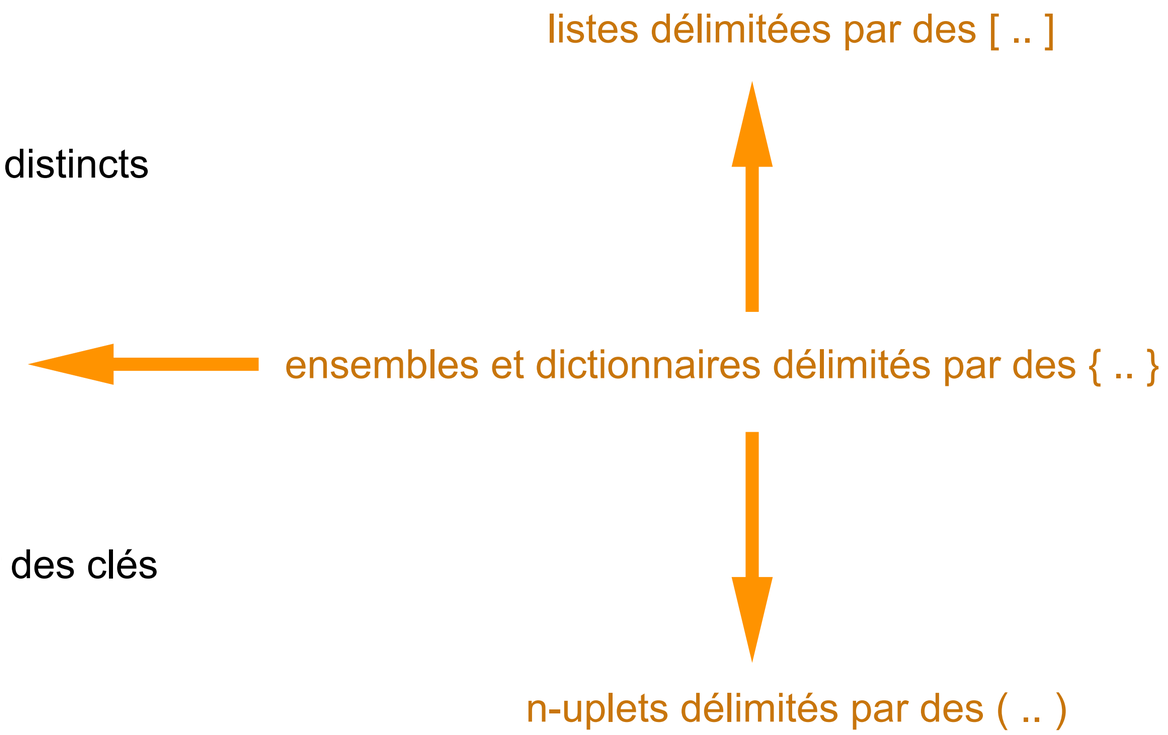
initialisation de la génération des nombres aléatoires

- la liste des modules disponibles s'obtient avec

```
help()
modules
. . .
quit
```

- la liste des fonctions disponibles : `help(random)`

Ensembles — Dictionnaires



- les **ensembles** sont des listes non ordonnées d'éléments tous distincts

```
print ({10, 2, 3} == {3, 2, 10})
True

print ({10, 2, 3} == {5, 2, 10})
False
```

- les **dictionnaires** sont des listes non ordonnées indexées par des clés

```
age = {"alice": 22, "bob": 27, "charlie": 30}

table1 = {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}

table2 = {1: 1, 2: 8, 3: 27, 4: 64, 5: 125, 6: 216, 7: 343}

print(age["bob"])

print(table1[5])

print(table2[6])
```

Compréhension

- on peut **générer** des tableaux, listes, ensembles, dictionnaires avec la notation compréhensive

```
a = [x**2 for x in range(21) if x*2 < 21]
print (a)
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

```
b = {2 * x for x in range (20)}
print (b)
{0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34, 36, 38}
```

```
c = {x : x**2 for x in range (10)}
print (c)
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6: 36, 7: 49, 8: 64, 9: 81}
```

```
def rand_array (n, m) :
    return [rd.randrange (m) for _ in range (n)]

for _ in range (4):
    print (rand_array (10, 100))
```

```
[63, 95, 66, 58, 47, 45, 60, 48, 99, 92]
[93, 8, 15, 60, 30, 15, 4, 43, 33, 98]
[71, 28, 65, 73, 93, 73, 35, 64, 95, 31]
[93, 13, 90, 31, 39, 32, 68, 62, 73, 69]
```

Tri par sélection *(selection sort)*

- on cherche le minimum et on le met en tête..
  et on recommence à partir du deuxième élément, etc…

```
def triSelection (a) :
    n = len (a)
    for i in range (n-1) :
        jmin = index_min_of (a, i)
        xchange (a, i, jmin)
```

échanger les valeurs de `a[i]` et `a[jmin]`

index du minimum dans `a` à partir de `i`

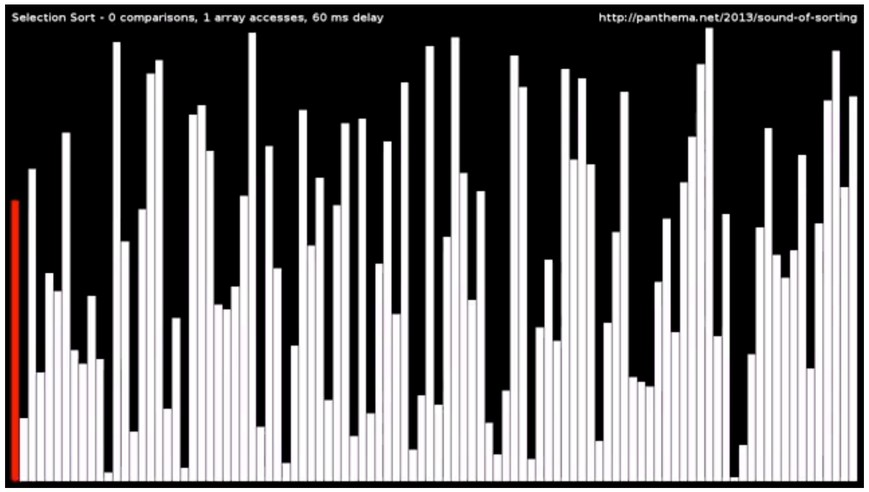
```
def index_min_of (a, i) :
    jmin = i
    for j in range (i+1, len(a)):
        if a[j] < a[jmin] :
            jmin = j
    return jmin
```

```
def xchange (a, i, j) :
    temp = a[i]
    a[i] = a[j]
    a[j] = temp
```

et en dépliant les fonctions:

```
def triSelection (a) :
    n = len (a)
    for i in range (n-1) :
        jmin = i
        for j in range (i+1, n):
            if a[j] < a[jmin] :
                jmin = j
        t = a[i]; a[i] = a[jmin]; a[jmin] = t
```

Exercice 5: générer une liste aléatoire et la trier



http://visualgo.net/en/sorting

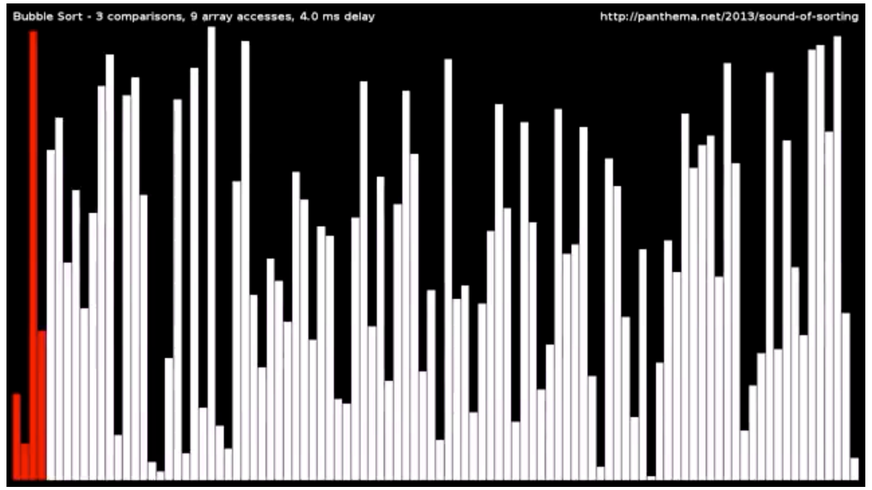
[texte des programmes]

Tri bulle *(bubble sort)*

- un bulle pousse le maximum vers la fin.. et
  on recommence jusqu'à l'avant-dernier élément, etc……

```
def triBulle (a) :
    n = len (a)
    for j in range (n-1, 0, -1) :
      for i in range (0, j):
        if a[i] > a [i+1] :
          t = a[i]; a[i] = a[i+1]; a[i+1] = t
```

échanger les valeurs de `a[i]` et `a[i+1]`



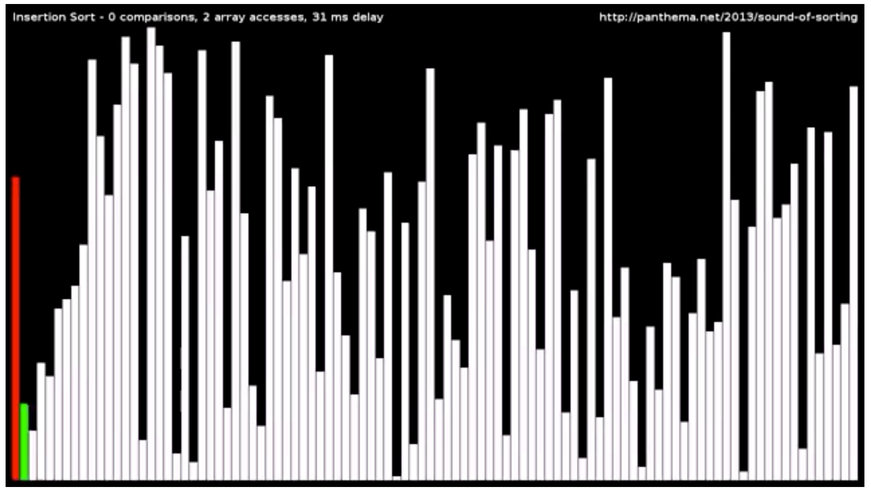
http://visualgo.net/en/sorting

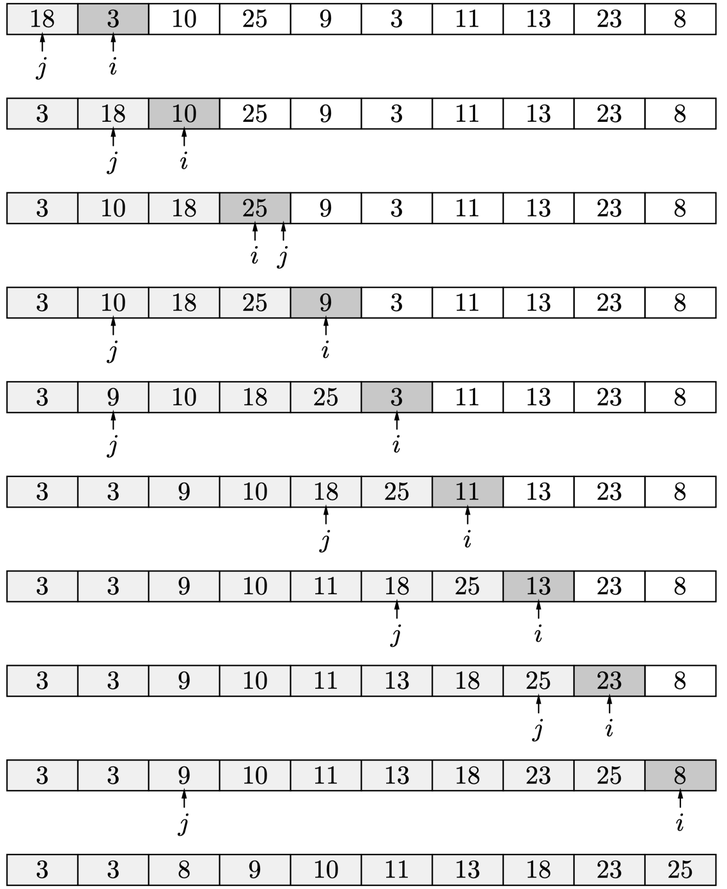
[texte des programmes]

Tri par insertion *(insertion sort)*

- on insère les éléments dans la partie gauche déjà triée, etc…… [comme dans un jeu de cartes]

```
def triInsertion (a) :
    n = len (a)
    for i in range (1, n) :
        v = a[i]; j = i
        while (j > 0 and a[j-1] > v) :
            a[j] = a[j-1];
            j = j - 1
        a[j] = v
```





http://visualgo.net/en/sorting

[texte des programmes]

Quelques remarques

- on peut trier des tableaux d'entiers, de réels, de chaînes de caractères

```
a = [44, 127, 24, 15, 60, 149, 147, 72, 36, 34]
b = [2.3, 2, 4.6]
c = [ 'camille', 'jean-jacques', 'paul', ‘axel']

d = [ 'camille', 28, 'paul', 'axel']
```

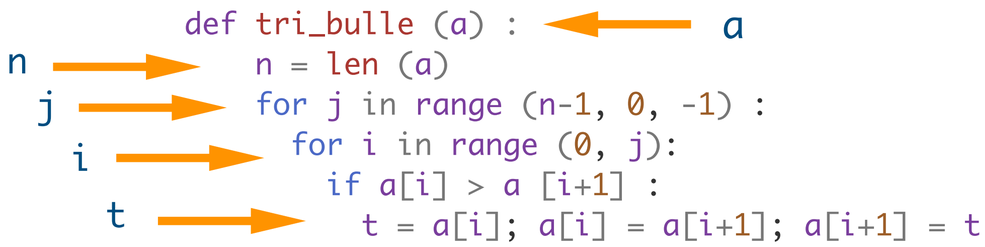
```
tri_selection (a)   →   [15, 24, 34, 36, 44, 60, 72, 127, 147, 149]
tri_selection (b)   →   [2, 2.3, 4.6]
tri_selection (c)   →   ['axel', 'camille', 'jean-jacques', 'paul']
```

```
tri_selection (d)   →   Traceback (most recent call last):
                          File "<stdin>", line 1, in <module>
                          File "<stdin>", line 6, in tri_selection
                        TypeError: '<' not supported between instances of 'int' and 'str'
```

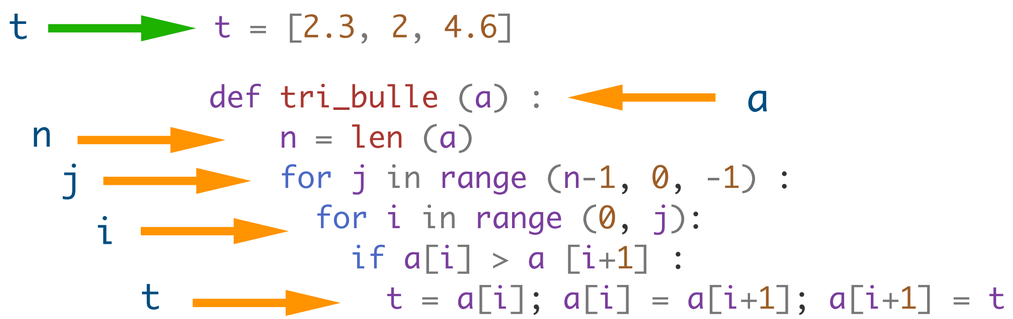
- en Python, les **types sont dynamiques**
- et on ne voit les erreurs de types qu'à l'exécution

Quelques remarques

- les variables déclarées dans une fonction n'existent que dans le code de cette fonction



- les variables `n`, `j`, `i`, `t` et *a* sont **locales** à la fonction `tri_bulle`



- la variable `t` **globale** est distincte de la variable `t` locale

Prochain cours

- réviser les classes et objets
- arbres
- recherche en table
- arbres binaires de recherche
- arbres équilibrés